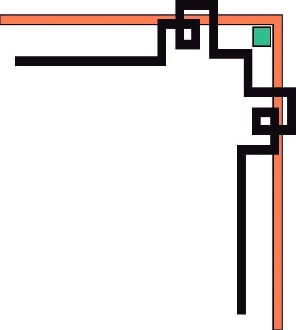
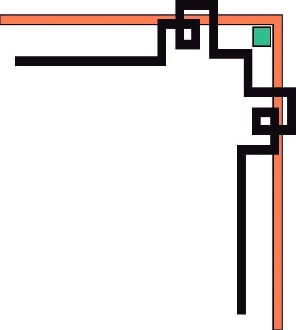
**ĐẠI HỌC BÁCH KHOA HÀ NỘI**



**VIỆN ĐIỆN TỬ - VIỄN THÔNG**

\_\_\_\_\_\_\_\_\_\_\_o\_0\_0\_\_\_\_\_\_\_\_\_\_\_\_

A picture containing chart

Description automatically generated

**BÁO CÁO BÀI TẬP LỚN**

**HỆ ĐIỀU HÀNH**

**Đề tài: Deadlock Detection Algorithm**

GVHD: TS. Hàn Huy Dũng

**Nhóm 5**

|  |  |
| --- | --- |
| Doãn Văn Thể | 20182797 |
| Nguyễn Thiện Đăng | 20182409 |
| Trần Văn Sỹ | 20182763 |

**Hà Nội, Tháng 2 năm 2022**

**Lời nói đầu**

Trong môi truờng đa chương, nhiều quá trình có thể cạnh tranh một số giới hạn  
tài nguyên. Một quá trình yêu cầu tài nguyên, nếu tài nguyên không sẳn dùng tại thời điểm đó, quá trình đi vào trạng thái chờ. Quá trình chờ có thể không bao giờ chuyển trạng thái trở lại vì tài nguyên chúng yêu cầu bị giữ bởi những quá trình đang chờ khác. Trường hợp này được gọi là deadlock (khoá chết).

Chúng ta sẽ mô tả các phương pháp mà hệ điều hành có thể dùng để ngăn chặn hay giải quyết deadlock. Hầu hết các hệ điều hành không cung cấp phương tiện ngăn chặn deadlock nhưng những đặc điểm này sẽ được thêm vào sau đó. Vấn đề deadlock chỉ có thể trở thành vấn đề phổ biến, xu hướng hiện hành gồm số lượng lớn quá trình, chương trình đa luồng, nhiều tài nguyên trong hệ thống và đặc biệt các tập tin có đời sống dài và những máy phục vụ cơ sở dữ liệu hơn là các hệ thống bó.

Chính vì thế, nhóm chúng em xin nghiên cứu đề tài “Deadlock Detection Algorithm ” trên hệ điều hành Linux dưới sự hướng dẫn của thầy Hàn Huy Dũng.

Ngoài phần mở đầu, báo cáo bao gồm 3 chương:

Chương 1: Tổng quan về Deadlock

Chương 2: Các giải thuật phòng tránh và xử lý Deadlock

Chương 3: Bài toán Philosophers

Trong quá trình tìm hiểu và thực hiện đề tài này, nhóm chúng em cũng đã dựa trên nhiều tài liệu khác nhau trên internet và do thày gợi ý. Bên cạnh đó kỹ năng và kiến thức chưa đủ sâu nên trong quá trình thực hiện cũng không tránh khỏi những sai sót, rất mong thầy góp ý, giúp đỡ thêm cho chúng em.

**MỤC LỤC**

**Lời nói đầu…………..…………………………….………..………......2**

**CHƯƠNG MỞ ĐẦU…………..………………….………..………......5**

***Tóm tắt báo cáo*…………………………………….………..……..….……..5**

***Phân công công việc*........................................................................................5**

**CHƯƠNG 1. TỔNG QUAN VỀ DEADLOCK…………………….........6**

***1.1. Giới thiệu về Deadlock*………………….……….….….……........................6**

***1.2. Ví dụ về Deadlock* ………………….……….….……………………...........7**

***1.3. Điều kiện xảy ra Deadlock*…………………………………………….…...10**

***1.4. Các phương án xử lý Deadlock*……………………………………………10**

**CHƯƠNG 2. CÁC GIẢI THUẬT PHÒNG TRÁNH VÀ XỬ LÝ DEADLOCK……………………………………………………………….…..13**

***2.1. Trạng thái an toàn*………………………………………………………….13**

***2.2. Giải thuật đồ thị cấp phát tài nguyên*……………………………………...13**

***2.3. Giải thuật Banker*……………………….………….…….……….………..14**

**CHƯƠNG 3. BÀI TOÁN DINING PHILOSOPHERS…………..…...18**

***3.1. Tổng quan bài toán*……………………………………………………..…..18**

***3.2. Sơ đồ thuật toán*…………………………………………………………….19**

***3.3. Thử nghiệm kết quả*……………..….………….…….……….………….....20**

***3.4. Ví dụ ngẫu nhiên*…………………………………………………………...24**

***3.5. Giải pháp cho bài toán*………………………………………………….….25**

***3.5.1. Xóa tiến trình*……………..….………….…….…….……………...26**

***3.5.2. Tiến trình ngủ*……………..….…….……….……………………...27**

***3.5.3. Trình tự sử dụng tài nguyên*……………..….………….…………..29**

**KẾT LUẬN………………………………………………………………………32**

**TÀI LIỆU THAM KHẢO………………………………………………………32**

**CHƯƠNG MỞ ĐẦU**

**Tóm tắt báo cáo**

Một deadlock xảy ra khi có một tập hợp các tiến trình đang chờ tài nguyên do các tiến trình khác trong cùng một tập hợp nắm giữ. Các quy trình trong deadlock chờ tài nguyên vô thời hạn và không bao giờ chấm dứt việc thực thi của chúng và tài nguyên chúng giữ không có sẵn cho bất kỳ quy trình nào khác. Sự xuất hiện của các bế tắc cần được kiểm soát một cách hiệu quả bằng cách phát hiện và giải quyết chúng, nhưng đôi khi có thể dẫn đến sự cố hệ thống nghiêm trọng. Bài báo cáo này mô tả phát hiện bế tắc bằng cách sử dụng thuật toán giải quyết bế tắc đồ thị cấp phát tài nguyên và giải thuật banker và giải quyết bài toàn Deadlock cơ bản Dining Philosophers.

**Phân công công việc**

***Bảng 1: Bảng phân công công việc***

|  |  |  |
| --- | --- | --- |
| STT | Thành Viên | Công việc cụ thể |
| 1 | Doãn Văn Thể (C) | Thực hiện chương trình, trình bày kết quả, nhận xét đánh giá kết quả, làm slide,báo cáo, thuyết trình |
| 2 | Nguyễn Thiện Đăng | Tổng hợp lý thuyết, tìm kiếm tài liệu, chạy chương trình, liệt kê so sánh kết quả, kiểm thử code, quản lý github |
| 3 | Trần Văn Sỹ | Tìm hiểu lý thuyết,nguyên lý hoạt động, tìm kiếm tài liệu, chạy chương trình, liệt kê so sánh kết quả |

**CHƯƠNG 1: TỔNG QUAN VỀ DEADLOCK**

**1.1 Giới thiệu về Deadlock**

Bế tắc là một tình huống trong đó một tập hợp các quy trình bị chặn bởi vì mỗi tiến trình đang giữ một tài nguyên và chờ một tài nguyên khác được thu nhận bởi một số tiến trình khác.

A picture containing text, book, vector graphics

Description automatically generated

Hãy xem xét một ví dụ khi một tên cướp đang giữ con tin còn cảnh sát đang giữ đồng bọn của tên cướp.Cả hai bên đều muốn đạt được mục đích riêng của mình tuy nhiên thì trong một thời gian ngắn,điều này sẽ khó mà có thể giải quyết được. Tình huống tương tự cũng xảy ra trong hệ điều hành khi có hai hoặc nhiều tiến trình giữ một số tài nguyên và chờ tài nguyên do tiến trình khác nắm giữ.

Diagram

Description automatically generated

Có một biến thể của deadlock được gọi là lovelock. Đây là tình huống trong đó hai hoặc nhiều quy trình liên tục thay đổi trạng thái của chúng để đáp ứng với những thay đổi trong các quy trình khác mà không thực hiện bất kỳ công việc hữu ích nào. Điều này tương tự như bế tắc ở chỗ không có tiến trình nào được thực hiện nhưng khác ở chỗ không quá trình nào bị chặn hoặc chờ đợi bất cứ điều gì. Một ví dụ của con người về sự sống động sẽ là hai người gặp mặt trực tiếp trong một cầu thang và mỗi người di chuyển sang một bên để người kia đi qua, nhưng cuối cùng họ lắc lư từ bên này sang bên kia mà không đạt được bất kỳ tiến bộ nào bởi vì họ luôn di chuyển theo cùng một cách tại cùng lúc.

Diagram, schematic

Description automatically generated

**1.2. Ví dụ về Deadlock**

Deadlock có thể xảy ra khi việc bạn sử dụng tài nguyên không đúng thứ tự hoặc không hợp lý.

Ví dụ 1:

Text

Description automatically generated

Ví dụ trên tài nguyên mutex thứ nhất bị khóa 2 lần liên tiếp

Ví dụ 2:

Text

Description automatically generated

Ở đây Thread 1 sau khi sử dung mutex đã không có quá trình giải phóng để các Thread khác tiếp tục thực hiện dẫn đến Deadlock

Ví dụ 3:

Text

Description automatically generated

Ở đây nhìn thì có vẻ ổn tuy nhiên hệ thống sẽ chỉ hoạt động bình thường trong micro giây đầu tiên.Thread1 sẽ giữ m1,Thread 2 giữ m2

Chart, line chart

Description automatically generated

Tới micro giây thứ 2 thì Thread thứ nhất sẽ yêu cầu gọi đến tài nguyên m2 trong khi Thread 2 đang nắm giữ m2 và tương tự ngược lại Thread 2 yêu cầu đến m1 trong khi m1 đang được giữ bởi Thread 1.Cả 2 Thread sẽ chờ đợi tài nguyên của nhau từ đó gây nên hiện tượng Deadlock.

Chính vì vậy việc đầu tiên cơ bản nhất để tránh deadlock đó là kiểm tra quy trình thứ tự sử dụng tài nguyên của mình trước khi cả thực hiện chương trình

Text

Description automatically generated

Kết quả:

Text

Description automatically generated with medium confidence

Thay đổi thứ tự sử dụng tài nguyên và vấn đề được giải quyết

**1.3. Điều kiện xảy ra Deadlock**

Có thể tránh được deadlock bằng cách tránh ít nhất một trong bốn điều kiện, bởi vì tất cả bốn điều kiện này được yêu cầu đồng thời để gây ra deadlock.

**Mutual Exclusion** :Các Tài nguyên loại trừ lẫn nhau được chia sẻ chẳng hạn như tệp chỉ đọc không dẫn đến bế tắc nhưng các tài nguyên, chẳng hạn như máy in và ổ đĩa băng, yêu cầu quyền truy cập độc quyền bởi một quy trình duy nhất.

**Hold and Wait:** Giữ và chờ Trong điều kiện này, các quy trình phải được ngăn chặn giữ một hoặc nhiều tài nguyên trong khi đồng thời chờ một hoặc nhiều tài nguyên khác.

**No Preemption** :Không có sự ưu tiên Việc lựa chọn phân bổ tài nguyên quy trình có thể tránh được tình trạng bế tắc, nếu có thể.

**Circular Wait** :Có thể tránh được thời gian chờ theo vòng tròn nếu chúng ta đánh số tất cả các tài nguyên và yêu cầu các quy trình chỉ yêu cầu tài nguyên theo thứ tự tăng (hoặc giảm) nghiêm ngặt.

**1.4. Các phương án xử lý vấn đề Deadlock**

Một tiến trình trong hệ điều hành sử dụng các tài nguyên khác nhau và sử dụng tài nguyên theo cách sau:

* Yêu cầu tài nguyên
* Sử dụng tài nguyên
* Giải phóng tài nguyên

Nói chung, có ba cách để xử lý các bế tắc:

**Ngăn chặn hoặc tránh deadlock** - Không để hệ thống rơi vào trạng thái deadlock, chúng ta có thể lấy một nguồn lực từ một quá trình và đưa nó cho quá trình khác. Điều này sẽ giải quyết tình trạng bế tắc, nhưng đôi khi nó gây ra vấn đề.

**Phát hiện và khôi phục deadlock** - Hủy bỏ một quy trình hoặc xử lý trước một số tài nguyên khi phát hiện thấy các deadlock. Ta sẽ loại bỏ một hoặc nhiều quy trình, đây là cách đơn giản nhất nhưng nó rất dễ thực hiện tuy nhiên đây không phải phương pháp tối ưu cho việc chúng ta muốn khắc phục lỗi.

**Bỏ qua tất cả vấn đề** - Nếu deadlock chỉ xảy ra mỗi năm một lần hoặc lâu hơn, có thể tốt hơn là chỉ để chúng xảy ra và khởi động lại nếu cần thiết hơn là phải chịu các hình phạt liên tục về hiệu suất hệ thống và chi phí liên quan đến việc phát hiện hoặc ngăn chặn deadlock. Đây là cách tiếp cận mà cả Windows và UNIX đều áp dụng.

Để tránh bế tắc, hệ thống phải có thêm thông tin về tất cả các quy trình. Đặc biệt, hệ thống phải biết những tài nguyên mà một tiến trình sẽ hoặc có thể yêu cầu trong tương lai. (Phạm vi từ mức tối đa trong trường hợp xấu nhất đơn giản đến yêu cầu tài nguyên hoàn chỉnh và kế hoạch phát hành cho mỗi quy trình, tùy thuộc vào thuật toán cụ thể.) Phát hiện deadlock khá đơn giản, nhưng khôi phục deadlock yêu cầu hủy bỏ quy trình hoặc sử dụng trước tài nguyên, cả hai đều không phải là một thay thế tối ưu. Nếu deadlock không được ngăn chặn cũng như không được phát hiện, thì khi xảy ra deadlock, hệ thống sẽ dần chậm lại, vì ngày càng có nhiều tiến trình bị mắc kẹt khi chờ đợi các tài nguyên hiện đang bị deadlock và các quy trình chờ đợi khác . Thật không may, sự chậm lại này có thể không phân biệt được với sự chậm lại của hệ thống nói chung khi một quy trình thời gian thực có nhu cầu tính toán cao.

Trong tính toán đồng thời, bế tắc là trạng thái trong đó mỗi thành viên của một nhóm đang chờ một thành viên khác, bao gồm cả chính mình, thực hiện hành động, chẳng hạn như gửi tin nhắn hoặc phổ biến hơn là giải phóng một khóa.

Diagram

Description automatically generated

Bế tắc là một vấn đề phổ biến trong các hệ thống đa xử lý, tính toán song song và hệ thống phân tán, nơi các khóa phần mềm và phần cứng được sử dụng để phân xử tài nguyên được chia sẻ và thực hiện đồng bộ hóa quy trình.

Trong hệ điều hành, deadlock xảy ra khi một tiến trình hoặc luồng đi vào trạng thái chờ vì tài nguyên hệ thống được yêu cầu được giữ bởi một tiến trình chờ khác, đến lượt nó lại chờ một tài nguyên khác do tiến trình chờ khác nắm giữ. Nếu một quy trình không thể thay đổi trạng thái của nó vô thời hạn bởi vì các tài nguyên được yêu cầu bởi nó đang được được sử dụng bởi một quá trình chờ đợi khác, khi đó hệ thống được cho là đang ở trong tình trạng bế tắc.

Trong một hệ thống thông tin liên lạc, sự cố tắc nghẽn xảy ra chủ yếu do tín hiệu bị mất hoặc bị hỏng chứ không phải do tranh chấp tài nguyên

**CHƯƠNG 2: GIẢI THUẬT PHÁT HIỆN VÀ PHÒNG TRÁNH DEADLOCK**

**2.1. Trạng thái an toàn**

Một trạng thái là an toàn nếu hệ thống có thể cấp phát các tài nguyên tới mỗi quá trình trong một vài thứ tự và vẫn tránh deadlock. Hay nói cách khác, một hệ thống ở trong trạng thái an toàn chỉ nếu ở đó tồn tại một thứ tự an toàn. Thứ tự của các quá trình<P1, P2, …, Pn>là một thứ tự an toàn cho trạng thái cấp phát hiện hành nếu đối với mỗi thứ tự Pi, các tài nguyên mà Pi yêu cầu vẫn có thể được thoả mãn bởi tài nguyên hiện có cộng với các tài nguyên được giữ bởi tất cả Pj, với j<i. Trong trường hợp này, nếu những tài nguyên mà quá trình Pi yêu cầu không sẳn dùng tức thì thì Pi có thể chờ cho đến khi tất cả Pj hoàn thành. Khi chúng hoàn thành, Pi có thể đạt được tất cả những tài nguyên nó cần, hoàn thành các tác vụ được gán, trả về những tài nguyên được cấp phát cho nó và kết thúc. Khi Pi kết thúc, Pi+1 có thể đạt được các tài nguyên nó cần,... Nếu không có thứ tự như thế tồn tại thì trạng thái hệ thống là không an toàn.

Graphical user interface, application, table

Description automatically generated

Để minh hoạ, chúng ta xét một hệ thống với 12 ổ băng từ và 3 quá trình: P­0, P1, P2. Quá trình P0 yêu cầu 10 ổ băng từ, quá trình P1 có thể cần 4 và quá trình P2 có thể cần tới 9 ổ băng từ. Giả sử rằng tại thời điểm t0, quá trình P0 giữ 5 ổ băng từ, quá trình P1 giữ 2 và quá trình P2 giữ 2 ổ băng từ. Tại thời điểm t0, hệ thống ở trạng thái an toàn. Thứ tự <P1,P0, P2> thoả điều kiện an toàn vì quá trình P1 có thể được cấp phát tức thì tất cả các ổ đĩa từ và sau đó trả lại chúng (sau đó hệ thống có 5 ổ băng từ sẳn dùng ), sau đó quá trình P0 có thể nhận tất cả ổ băng từ và trả lại chúng (sau đó hệ thống sẽ có 10 ổ băng từ sẳn dùng), và cuối cùng quá trình P2 có thể nhận tất cả ổ băng từ của nó và trả lại chúng (sau đó hệ thống sẽ có tất cả 12 ổ băng từ sẳn dùng).

**2.2. Giải thuật đồ thị cấp phát tài nguyên**

Nếu chúng ta có một hệ thống cấp phát tài nguyên với một thể hiện của mỗi loại, một biến dạng của đồ thị cấp phát tài nguyên có thể được dùng để tránh deadlock.Nó đưa hệ thống về trạng thái sử dụng tài nguyên an toàn đã trình bày ở phần trước.Mô tả đơn giản theo hình vẽ là thay vì cả 2 tiến trình P1 và P2 cùng đang yêu cầu và đợi tài nguyên của mình thì ta sẽ dùng một tiến trình trung gian thứ 3 để thực hiện các công việc này.Hai tiến trình P1 P2 sẽ lần lượt thực hiện công việc của mình mà không giàng buộc bất cứ tài nguyên nào bởi tiến trình khác và sau khi hoàn thành nó sẽ trả về tài nguyên để tiến trình thứ 3 thực hiện.

Chart

Description automatically generated with medium confidence

**2.3. Giải thuật Banker**

Giải thuật đồ thị cấp phát tài nguyên không thể áp dụng tới hệ thống cấp phát tài nguyên với nhiều thể hiện của mỗi loại tài nguyên. Giải thuật tránh deadlock mà chúng ta mô tả tiếp theo có thể áp dụng tới một hệ thống nhưng ít hiệu quả hơn cơ chế đồ thị cấp phát tài nguyên. Giải thuật này thường được gọi là giải thuật của Banker. Tên được chọn vì giải thuật này có thể được dùng trong hệ thống ngân hàng để đảm bảo ngân hàng không bao giờ cấp phát tiền mặt đang có của nó khi nó không thể thoả mãn các yêu cầu của tất cả khách hàng.

Khi một quá trình mới đưa vào hệ thống, nó phải khai báo số tối đa các thể hiện của mỗi loại tài nguyên mà nó cần. Số này có thể không vượt quá tổng số tài nguyên trong hệ thống. Khi một người dùng yêu cầu tập hợp các tài nguyên, hệ thống phải xác định việc cấp phát của các tài nguyên này sẽ để lại hệ thống ở trạng thái an toàn hay không. Nếu trạng thái hệ thống sẽ là an toàn, tài nguyên sẽ được cấp; ngược lại quá trình phải chờ cho tới khi một vài quá trình giải phóng đủ tài nguyên.

Nhiều cấu trúc dữ liệu phải được duy trì để cài đặt giải thuật Banker. Những cấu trúc dữ liệu này mã hoá trạng thái của hệ thống cấp phát tài nguyên. Gọi n là số quá trình trong hệ thống và m là số loại tài nguyên trong hệ thống. Chúng ta cần các cấu trúc dữ liệu sau:

* Available: một vector có chiều dài m hiển thị số lượng tài nguyên sẳn dùng của mỗi loại. Nếu Available[j]= k, có k thể hiện của loại tài nguyên Rj sẳn dùng.
* Max: một ma trận n x m định nghĩa số lượng tối đa yêu cầu của mỗi quá trình. Nếu Max[ i , j ] = k, thì quá trình Pi có thể yêu cầu nhiều nhất k thể hiện của loại tài nguyên Rj.
* Allocation: một ma trận n x m định nghĩa số lượng tài nguyên của mỗi loại hiện được cấp tới mỗi quá trình. Nếu Allocation[ i, j ] = k, thì quá trình Pi hiện được cấp k thể hiện của loại tài nguyên Rj.
* Need: một ma trận n x m hiển thị yêu cầu tài nguyên còn lại của mỗi quá trình. Nếu Need[ i, j ] = k, thì quá trình Pi có thể cần thêm k thể hiện của loại tài nguyên Rj để hoàn thành tác vụ của nó. Chú ý rằng, Need[ i, j ]= Max[ i, j ] – Allocation [ i, j ].

**Mô hình cấp phát và sử dụng tài nguyên của hệ thống**

Diagram

Description automatically generated

Diagram

Description automatically generated

**Tính toán tài nguyên cần dùng**

Table

Description automatically generated

Từ bảng bên trái dựa vào giá trị MAX và Allocation đã biết ta sẽ tính được giá trị Need cần thiết, sau đó so sánh Need với Available để xem có đủ điều kiện để tiến trình hoạt động hay không

Ở đây ta thấy, với process P0 tính được Need(7,4,3) > Available(3,3,2)

Chính vì thế P0 sẽ chưa được thực hiện, lần lượt xuống P1 P2 P3 khảo sát tương tự như vậy

**Kết quả thực hiện**

Text

Description automatically generated

Đưa vào lượng tài nguyên để mô phỏng giải thuật,trường hợp này có 4 loại tài nguyên và 4 tiến trình tạo ra ,lượng tài nguyên đưa vào là 1 2 0 1 khá nhỏ so với nhu cầu MAX để tiến trình thực hiện và cho kết quả Deadlock ở P1 P2 P3

Text

Description automatically generated with low confidence

Trong một trường hợp khác khi đưa vào lượng tài nguyên đủ lớn thì hệ thống không phát hiện deadlock ở các tiến trình

**CHƯƠNG 3: BÀI TOÁN DINING PHILOSOPHER**

**3.1.Tổng quan bài toán**

Bữa tối của các triết gia (dining philosophers problem) là một ví dụ nổi tiếng khi nhắc đến các vấn đề trong bài toán xử lý concurrent.

Vấn đề được phát biểu như sau: Cho 5 triết gia ngồi chung một bàn tròn với 5 chiếc đũa xếp xem kẽ giữa 2 người ngồi cạnh nhau như hình

A picture containing decorated

Description automatically generated

Mỗi triết gia tìm cách để ăn được thức ăn từ đĩa của mình với điều kiện: “chỉ ai có 2 chiếc đũa cạnh mình mới được phép ăn”, do đó họ lần lượt đổi trạng thái giữa ăn (eating) và đợi (thinking) 😊) Mỗi người sau khi giữa đôi đũa để ăn sau 1 khoảng thời gian phải bỏ lại 2 chiếc đũa về vị trí cũ để tiếp tục quá trình này. Yêu cầu: tìm một phương pháp đảm bảo để các triết gia đều có thể được ăn / đợi đổi lượt để không ai bị chết đói (chỉ đợi chứ không được ăn).

**Các trường hợp xảy ra:**

Trường hợp 1 : lần lượt từng người lấy đũa,tiến hành ăn,ăn xong,đặt đũa xuống sau đó tới người tiếp theo.Như vậy sẽ không có bế tắc nào xảy ra ở đây.

Trường hợp thứ 2: tất cả các triết gia đều cầm đũa của mình và chờ đũa từ người bên cạnh để có thể ăn:

Text

Description automatically generated with low confidence

Deadlock sẽ xảy ra ở đây khi không ai sẽ có thể ăn

Vậy giải pháp để không có triết gia nào bị đói là gì?

**3.2. Sơ đồ thuật toán**

**Shape

Description automatically generated with medium confidence**

Bắt đầu chương trình sẽ khởi tạo các thư viện và lượng tài nguyên cần dùng,định nghĩa số lượng các triết gia và thức ăn tương ứng với các luồng tạo ra và lượng công việc sẽ thực hiện.Tạo ra n Thread cho chúng hoạt động đồng thời,đưa các Thread vào vòng lặp,ban đầu sẽ kiểm tra số lượng thức ăn còn lại trên bài,check id tương ứng với 1 người sau đó người đó sẽ cầm đũa,nếu cầm đủ sẽ thực hiện ăn,lượng thức ăn sẽ giảm đi 1,sau khi ăn xong sẽ bỏ đũa xuống,tiếp tục quay lại kiểm tra lượng thức ăn còn lại trên bàn cho đến khi lượng thức ăn còn lại =0 sẽ kết thúc chương trình.

**3.3. Thử nghiệm kết quả**

Tạo ra 5 tiến trình và lượng thức ăn ban đầu là 50.

Text

Description automatically generated

Kết quả thử nghiệm chương trình cho thấy Deadlock có thể xảy ra một cách ngẫu nhiên,khả năng Deadlock và thời gian thực hiện sẽ tùy thuộc vào hiệu năng và khả năng lập lịch của mutilthreading trên mỗi máy tính.

Graphical user interface, application, table

Description automatically generated

*Bảng 1: Khảo sát khả năng Deadlock của chương trình*

Thực hiện khảo sát đo 25 lần với thông số tạo ra ban đầu là 5 người và 50 thức ăn ta thu được kết quả như bảng 1.Ta thấy có 7/25 lần Deadlock tương đương 28% và thời gian thực thi chương trình khi không có Deadlock trung bình khoảng 6.206 s

1. **Thay đổi số luồng ban đầu tạo ra**

Tăng số tiến trình lên 6,giữ nguyên lượng thức ăn là 50

Table

Description automatically generated

*Bảng 1: Khảo sát khả năng Deadlock của chương trình khi số luồng lên 6*

Kết quả cho thấy có 2/25 lần chương trình bị Deadlock .Vậy khi tăng số luồng thực hiện lên và giữ nguyên lượng thức ăn ta nhận thấy tỉ lệ Deadlock trong các lần đo giảm đi và thời gian thực hiện chương trình cũng giảm.Thời gian trung bình khi tăng số luồng lên 6 là 4.838 s.

Tương tự như vậy,đến khi tăng số luồng lên 10,không còn thấy hiện tượng Deadlock xảy ra,thời gian thực thi chỉ khoảng 2.1556 s

Table

Description automatically generated

*Bảng 1: Khảo sát khả năng Deadlock của chương trình khi số luồng lên 10*

Và bảng thống kê số lần Deadlock và thời gian thực thi khi khảo sát từ 5-> 10 luồng:

|  |  |  |
| --- | --- | --- |
| Số tiến trình | Thời gian thực thi TB(s) | Số lần Deadlock |
| 5 | 6.206 | 6 |
| 6 | 4.838 | 4 |
| 7 | 3.554 | 2 |
| 8 | 2.969 | 2 |
| 9 | 2.206 | 1 |
| 10 | 1.733 | 0 |

*Bảng 2: Bảng thống kê số lần Dealock và thời gian thực hiện khi tăng số tiến trình*

Chart, histogram

Description automatically generated

*Hình 5: Biểu đồ thời gian và số lần Deadlock phụ thuộc vào số tiến trình tạo ra*

1. **Thay đổi lượng thức ăn ban đầu**

Giữ nguyên số tiến trình là 5

Thay đổi lượng thức ăn ban đầu từ 5 đến 100

Graphical user interface, text

Description automatically generated

* Deadlock vẫn xảy ra một cách ngẫu nhiên,tuy nhiên khi tăng lượng thức ăn lên thì tỉ lệ Deadlock cũng tăng lên,vì khi có nhiều công việc cần thực hiện hơn thì khả năng xung đột của các tiến trình cũng sẽ tăng lên và thời gian thực hiện chương trình cũng sẽ tăng theo.

|  |  |  |  |
| --- | --- | --- | --- |
| Số tiến trình | Số thức ăn | Thời gian thực thi TB(s) | Số lần Deadlock |
| 5 | 5 | 0.045 | 0 |
| 5 | 10 | 0.25 | 2 |
| 5 | 20 | 1.042 | 2 |
| 5 | 40 | 4.102 | 3 |
| 5 | 50 | 6.206 | 6 |
| 5 | 60 | 8.57 | 12 |
| 5 | 80 |  | 25 |

*Bảng 3: Bảng thống kê số lần Dealock và thời gian thực hiện khi tăng số thức ăn ban đầu*

Graphical user interface, chart

Description automatically generated

*Hình 5: Biểu đồ thời gian và số lần Deadlock phụ thuộc vào số công việc tạo ra*

**3.4.** **Ví dụ ngẫu nhiên**

Text

Description automatically generated with medium confidence

Tăng số luồng lên 10 và lượng thức ăn lên 1000,số công việc thực hiện được dao động khoảng 200 và thời gian khoảng 1 phút

Text

Description automatically generated

Tăng số tiến trình lên 100 và lượng thức ăn lên 5000,Deadlock xảy ra ngẫu nhiên theo từng thời điểm,có thể hoàn thành trong vòng 71 phút hoặc Deadlock trong 13ph tùy thuộc vào từng máy.

Text

Description automatically generated

**3.5. Giải pháp cho bài toán**

**3.5.1. Xóa một tiến trình**

Diagram, shape

Description automatically generated

* Xóa P5 ra khỏi bàn
* Khi đó P1 sẽ có thể cầm được cả 2 đũa trái phải và thực hiện ăn
* Sau khi ăn xong P1 sẽ bỏ đũa xuống cho P2 tiếp tục thực hiện và lần lượt đến P3,P4 tới khi thức ăn không còn

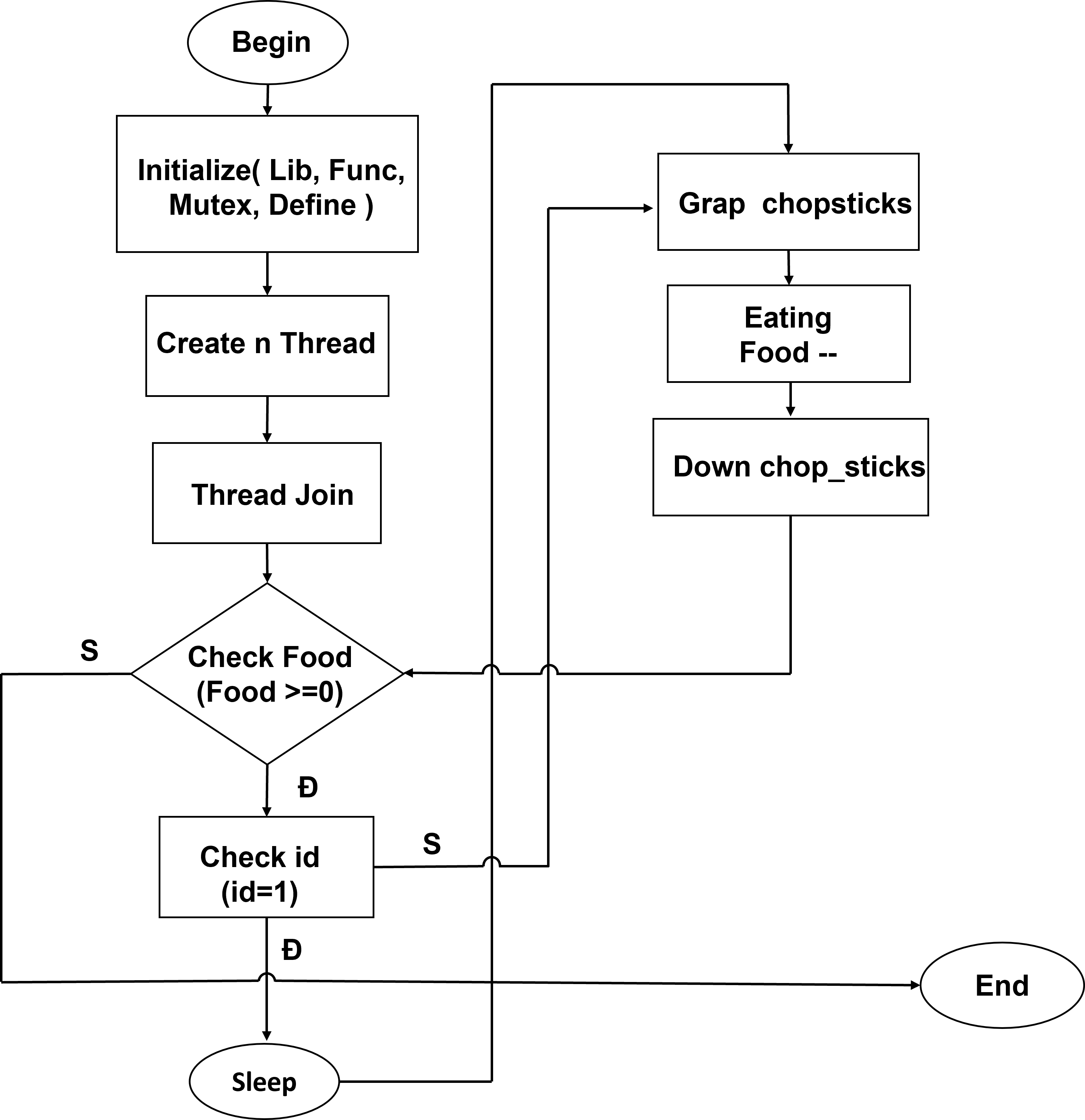
A picture containing text

Description automatically generated

Khi thực hiện giải pháp trên, chương trình không còn gây ra Deadlock,việc tăng số lượng tài nguyên chỉ làm tăng thời gian xử lý của hệ thống. Tuy nhiên nó có thể giải quyết vấn đề Deadlock nhưng không phải điều chúng ta mong muốn.Mỗi tiến trình chúng ta tạo ra đều có mục đích và có một nhiệm vụ khác nhau,không thể dễ dàng xóa một tiến trình đi trong một chương trình và điều chúng ta hướng đến là tất cả các tiến trình tạo ra đều được thực hiện(Tất cả mọi người đều không bị đói)

**3.5.2. Tiến trình ngủ**

* Đưa ít nhất 1 người vào trạng thái sleep trước khi họ tìm được đũa của mình
* Trong khoảng thời gian đó những ngườ khác sẽ thực hiện công việc của mình

****

* Sau khi người đưa vào trạng thái ngủ tỉnh dạy,họ sẽ tiếp tục thực hiện công việc
* Yêu cầu thời gian ngủ đủ lâu

A picture containing graphical user interface

Description automatically generated

Tùy thuộc vào thời gian ngủ chúng ta đặt,Deadlock vẫn có thể xảy ra

Bảng thống kê thời gian thực thi và số lần Deadlock khi chúng ta tăng thời gian ngủ của 1 tiến trình.

|  |  |  |
| --- | --- | --- |
| Thời gian ngủ | Thời gian thực thi TB(s) | Số lần Deadlock |
| 1 | 5.177 | 3 |
| 2 | 4.616 | 2 |
| 3 | 3.834 | 2 |
| 4 | 4.069 | 0 |
| 5 | 5.011 | 0 |
| 6 | 6.012 | 0 |
| 7 | 7.011 | 0 |
| 8 | 8.012 | 0 |

*Bảng 4: Bảng thống kê số lần Dealock và thời gian thực hiện khi tăng slee\_thread*

Chart, line chart

Description automatically generated

*Hình 6: Biểu đồ thời gian và số lần Deadlock phụ thuộc vào thời gian ngủ*

**3.5.3. Thay đổi thứ tự sử dụng tài nguyên**

Thay vì như thông thường mỗi người sẽ cầm 1 đũa bên phải của mình sau đó mới tìm tới đũa của người bên cạnh để thực hiện ăn. Chúng ta Cho một người (P5) sẽ cầm đũa bên trái trước(Đũa 1) khi đó cả 2 người P1 và P5 sẽ cùng giành lấy Đũa 1 thời điểm ban đầu. Một trong 2 người sẽ cầm được đũa 1 trước, và khi đó người còn lại không có đũa sẽ rơi vào trạng thái ngủ,không thể thực hiện thao tác cầm đũa thứ 2 khi đũa thứ nhất còn chưa có. Một tiến trình dừng hoạt động,các tiến trình còn lại lần lượt thực hiện bình thường cho đến khi tiếp tục vòng tiếp theo P1 hoặc P5 đã ăn xong và bỏ đũa xuống.

Diagram

Description automatically generated

Deadlock sẽ được giải quyết và tất cả các tiến trình đều có thể được thực hiện công việc của mình.

***Kết quả:***

A picture containing text

Description automatically generated

Đã giải quyết được vấn đề Deadlock,khi tăng lượng FOOD lên con số lớn hệ thống vẫn hoạt động bình thường

***Giải quyết hạn chế của phương pháp trước:***

Text

Description automatically generated with medium confidence

Vẫn có sự chênh lệch tuy nhiên nó đã đồng đều hơn khá nhiều và sự chênh lệch không quá lớn

Table

Description automatically generated

Chart, line chart

Description automatically generated

Thời gian thực thi của các tiến trình trong cùng một hệ thống là khác nhau,có sự chênh lệch và thay đổi ngẫu nhiên trong mỗi lần đo,phụ thuộc vào hiệu năng của máy và cách thức hoạt động của bộ lập lịch Multil Threading của hệ thống đó.

**KẾT LUẬN**

Qua bài nghiên cứu trên, chúng em đã tìm hiểu được về cơ chế Deadlock của một chương trình, khái niệm, điều kiện khả năng xảy ra, phương pháp khắc phục và nghiên cứu tìm hiểu bài toán thực tế Dining Philosophers. Đo đạc được các thông số cần thiết, chứng mình được hệ thống có deadlock và sự phụ thuộc của nó vào các yếu tố xung quanh. Trong quá trình tìm hiểu và thực hiện đề tài này, nhóm chúng em cũng đã dựa trên nhiều tài liệu khác nhau và nhóm em cảm ơn thầy đã hướng dẫn nhóm trong quá trình lựa chọn và hoàn thiện đề tài bài tập lớn. Bên cạnh đó kỹ năng và kiến thức chưa đủ sâu nên trong quá trình thực hiện chúng em cũng không tránh khỏi những sai sót, rất mong thầy góp ý, đưa ra nhận xét để giúp nhóm cải thiện thêm vào lần sau.

**TÀI LIỆU THAM KHẢO**

[**https://github.com/DwangDNguyen/deadlock/tree/master**](https://github.com/DwangDNguyen/deadlock/tree/master) **- link Github**

<https://manualzz.com/doc/26226325/deadlock-detection-and-recovery-in-linux>

<https://vi.wikipedia.org/wiki/Deadlock> - Tìm hiểu về Deadlock

<https://www.slideshare.net/omairimtiaz10/circularwait-operating-systems> - Tìm hiểu về Circular Wait

<https://www.cs.rpi.edu/academics/courses/fall04/os/c10/> - Operating Systems DeadLock

<https://www.jobilize.com/online/course/0-7-deadlock-h-i-u-hanh-by-openstax?page=9> – Giải thuật Banker

<https://docs.oracle.com/cd/E19205-01/820-0619/geosb/index.html> - Dining Philosophers

https://www.ijert.org/an-overview-on-deadlock-resolution-techniques